std::accumulate

Defined in header <numeric>

```
template< class InputIt, class T >
T accumulate( InputIt first, InputIt last, T init );                    (1)

template< class InputIt, class T, class BinaryOperation >
T accumulate( InputIt first, InputIt last, T init,                      (2)
              BinaryOperation op );
```

Computes the sum of the given value `init` and the elements in the range `[first, last)`. The first version uses `operator+` to sum up the elements, the second version uses the given binary function `op`.

op must not have side effects.	(until C++11)
op must not invalidate any iterators, including the end iterators, or modify any elements of the range involved.	(since C++11)

Parameters

first, last - the range of elements to sum

init - initial value of the sum

op - binary operation function object that will be applied. The binary operator takes the current accumulation value a (initialized to `init`) and the value of the current element b.

The signature of the function should be equivalent to the following:

```
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`.
The type `Type1` must be such that an object of type `T` can be implicitly converted to `Type1`. The type `Type2` must be such that an object of type `InputIt` can be dereferenced and then implicitly converted to `Type2`. The type `Ret` must be such that an object of type `T` can be assigned a value of type `Ret`.

Type requirements

- `InputIt` must meet the requirements of `InputIterator`.
- T must meet the requirements of `CopyAssignable` and `CopyConstructible`.

Return value

1) The sum of the given value and elements in the given range.
2) The result of left fold of the given range over op

Notes

Although `std::accumulate` performs left fold by default, right fold may be achieved by using reverse iterators, e.g. `std::accumulate(v.rbegin(), v.rend(), init, binop)`

Possible implementation

First version

```
template<class InputIt, class T>
T accumulate(InputIt first, InputIt last, T init)
{
    for (; first != last; ++first) {
        init = init + *first;
    }
    return init;
}
```

Second version

```
template<class InputIt, class T, class BinaryOperation>
T accumulate(InputIt first, InputIt last, T init,
             BinaryOperation op)
{
    for (; first != last; ++first) {
        init = op(init, *first);
    }
    return init;
}
```

Example

Run this code

```
#include <iostream>
#include <vector>
#include <numeric>
#include <string>
#include <functional>

int main()
{
    std::vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    int sum = std::accumulate(v.begin(), v.end(), 0);

    int product = std::accumulate(v.begin(), v.end(), 1, std::multiplies<int>());

    std::string s = std::accumulate(v.begin(), v.end(), std::string{},
                                    [](const std::string& a, int b) {
                                        return a.empty() ? std::to_string(b)
                                               : a + '-' + std::to_string(b);
                                    });

    std::cout << "sum: " << sum << '\n'
              << "product: " << product << '\n'
              << "dash-separated string: " << s << '\n';
}
```

Output:

```
sum: 55
product: 3628800
dash-separated string: 1-2-3-4-5-6-7-8-9-10
```

See also

adjacent_difference	computes the differences between adjacent elements in a range (function template)
inner_product	computes the inner product of two ranges of elements (function template)
partial_sum	computes the partial sum of a range of elements (function template)
reduce (C++17)	similar to **std::accumulate**, except out of order (function template)